dAV1d

dav1d is an AV1 (video) decoder

Jean-Baptiste Kempf, Videolabs / Videolan
Ronald S. Bultje, Two Orioles / FFmpeg

#VDD2018, sponsored by Google

dav1d project is sponsored by the AOM Alliance

dav1d: introduction

- AV1 is the new video codec by the Alliance for Open Media (AOM)
  - 20-30% lower bandwidth for the same quality content
  - Intended users are internet video streaming services (Netflix, Youtube, etc.)
  - There is currently only one decoder: libaom - we want to fix that
- Goals of our project:
  - Smaller binary size
  - Smaller memory footprint
  - Lower CPU usage

  } (*vs. libaom*)
  - Multi-threaded
  - Cross-Platform
  - Maintainable & Easy to understand
  - Well-integrated in downstream tools / applications
  - Libre & actually Open Source

dav1d: the fine prints

BSD license

- Unusual for us
- Same as libvorbis, opus
  RMS approved
- We want forks
  (notably for hardware people)
- We want everyone to ship it
  (including OS and browsers)
- Outside of FFmpeg, yet easy to integrate
  (Simple API)

VideoLAN project

Technical details

- C99
  - No VLA, No Complex
  - No GNU extension
- ASM
  - No intrinsics
  - ASM files, like in x264 and FFmpeg
- Buildsystem
  - Meson, ninja
  - MSVC, Xcode…
- Tools
  - C or Rust
  - MFC in C++

Footprint (= size) *

	dav1d	libaom
Source (kLOC)	28.5	262.8
Binary (MB)	0.64	1.73
Memory (MB) **	42.8	162.9

→ Smaller source code

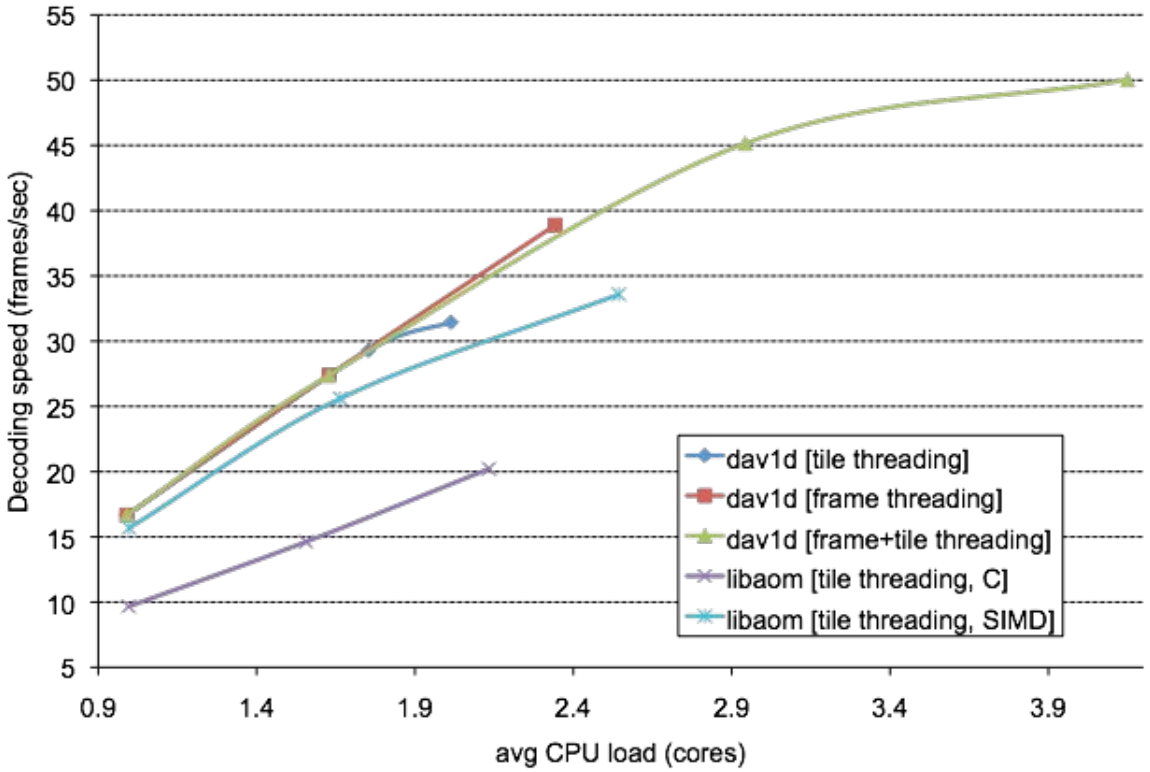
→ Smaller binary executable

→ Smaller runtime memory footprint

** av1/encoder in libaom excluded, SIMD excluded (*.[ch] only), built using -DCONFIG_AV1_ENCODER=0 -DAOM_TARGET_CPU=generic*

*** when playing http://download.opencontent.netflix.com.s3.amazonaws.com/AV1/Chimera/Chimera-AV1-8bit-1920x1080-6736kbps.ivf*

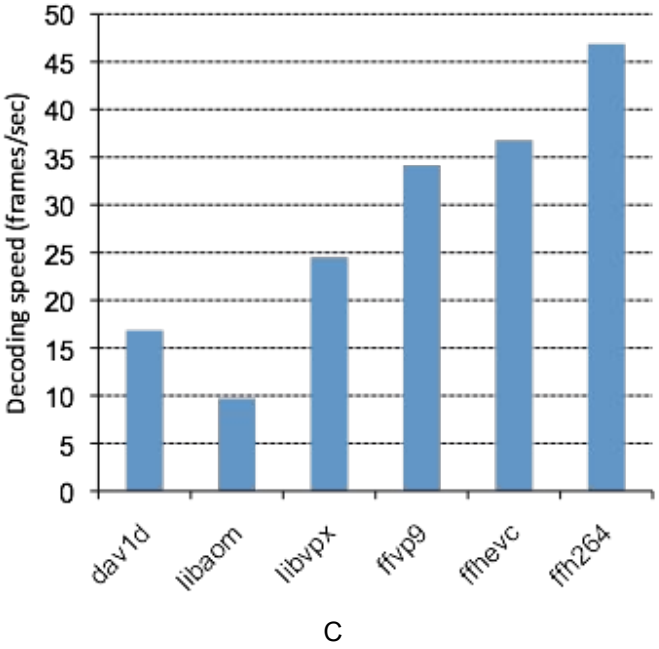
CPU usage (= speed)



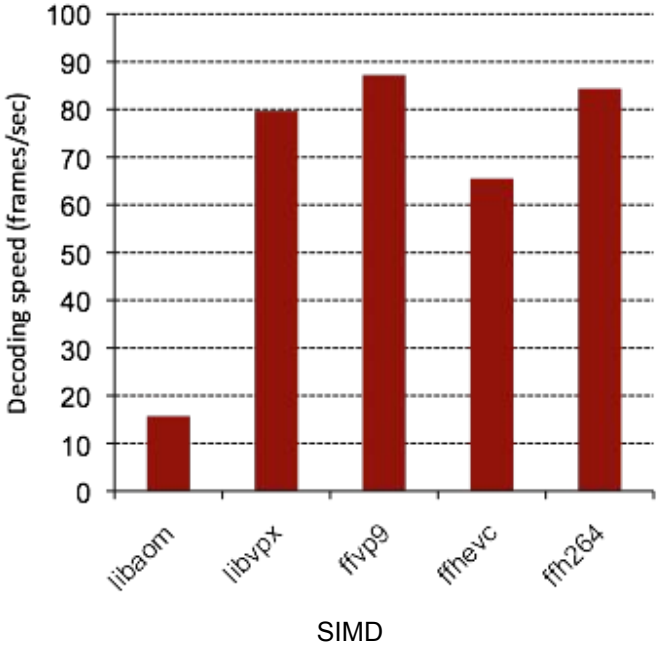
	Threads Frame x Tile	Time (sec) * Real	User
dav1d	1	59.982	59.397
	2 x 1	36.522	59.48
	4 x 1	25.709	60.238
	1 x 2	34.143	59.877
	1 x 4	31.442	63.326
	2 x 2	22.151	65.182
	4 x 4	19.989	82.961
libaom (C)	1	103.502	103.135
	2	68.354	106.412
	4	49.472	105.553
libaom (SIMD)	1	63.788	63.622
	2	39.066	64.998
	4	29.776	75.744

** when playing 1000 frames of http://download.opencontent.netflix.com.s3.amazonaws.com/AV1/Chimera/Chimera-AV1-8bit-1920x1080-6736kbps.ivf*

Speed compared to other video codecs



C



SIMD

	Decoder	Time (sec) *
C	dav1d	59.397
	libaom	103.135
	libvpx	40.817
	ffvp9	29.305
	ffhevc	27.203
	ffh264	21.339
SIMD	libaom	63.622
	libvpx	12.551
	ffvp9	11.463
	ffhevc	15.265
	ffh264	11.853

** when playing 1000 frames (single-threaded) of Chimera-AV1-8bit-1920x1080 compressed using ffmpeg -c:v lib{x264/x265/vpx-vp9} @ 4mbps*

High-level overview of decoding process (1)

main()

```
dav1d_init();
Dav1dContext *c;
dav1d_open(&c, ..);
for (;;) {
    // read data (e.g. from file)
    Dav1dData *in = ..;
    Dav1dPicture pic = { 0 };
    dav1d_decode(c, data, &pic);
    // do something with output pic
    ..
    dav1d_picture_unref(&pic);
}
dav1d_close(c);
```

dav1d_decode() → parse_obus()

```
for (;;) {
    obu_type = ..;
    switch (obu_type) {
    case OBU_SEQ_HDR:
        parse_seq_hdr(..);
        Break;
    case OBU_FRAME:
    case OBU_FRAME_HDR:
        parse_frame_hdr(..);
        if (obu_type == OBU_FRAME_HDR)
            break;
    case OBU_TILE_GRP:
        parse_tile_hdr(..);
        break;
    }
    if (full frame available)
        submit_frame(..);
}
```

High-level overview of decoding process (2)

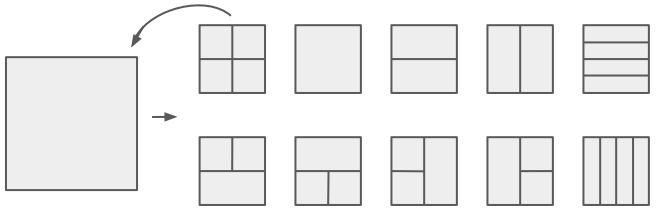
(main thread) (frame thread) (tile thread)

submit_frame() → **decode_frame()** → **decode_tile_sbrow()**

```
// setup tile data structures
for (int row = 0; row < hdr->tile_rows; row++)
    for (int col = 0; col < hdr->tile_cols; col++)
        tile_data[tile_idx++] = ..

// decode tiles (block parsing & reconstruction)
for (int pass = use_2pass; pass <= 2 * use_2pass;
     pass++)
{
    for (int row = 0; row < hdr->tile_rows; row++) {
        for (int sby = hdr->sb_row[row];
             sby < hdr->sb_row[row + 1]; sby++)
        {
            for (int col = 0; col < hdr->tile_cols;
                 col++)
            {
                decode_tile_sbrow(..);
            }
            postfilter_sbrow(..);
        }
    }
}
```

```
for (int sbx = hdr->sb_col[col];
     sbx < hdr->sb_col[col + 1]; sbx++)
{
    decode_sb(.., seqhdr->sb128 ? BL_128X128 : BL_64X64);
}
```



decode_sb()

```
enum BlockPartition bp = ..
if (bl != BL_8X8 && bp == PARTITION_SPLIT) {
    for (int n = 0; n < 4; n++)
        decode_sb(.., bl + 1);
} else {
    for (int n = 0; n < n_blks[bp]; n++)
        decode_b(.., blk_sz[bl][bp][n]);
}
```

High-level overview of decoding process (3)

`decode_b()`, pass != 2, symbol parsing

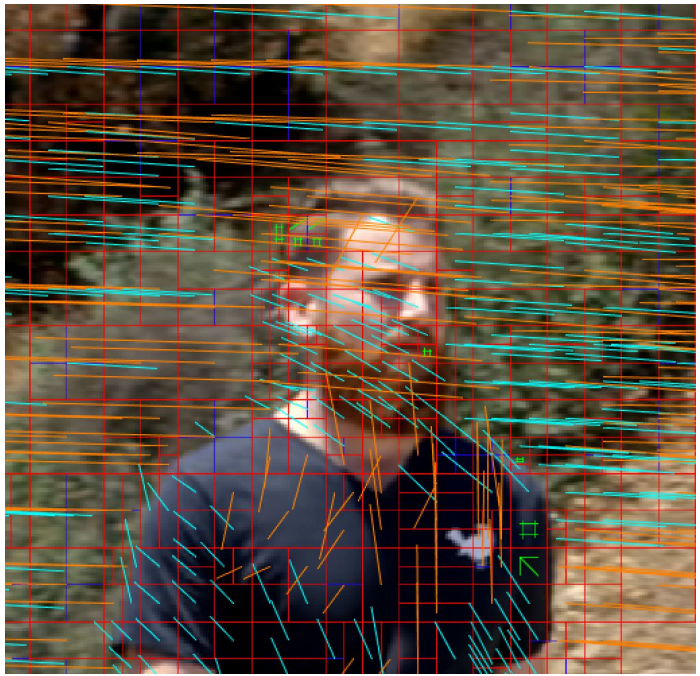
```
typedef struct Av1Block {
    uint8_t bl, bs, bp;
    uint8_t intra, seg_id, skip_mode, skip, uvtx;
    union {
        struct {
            uint8_t y_mode, uv_mode, tx, pal_sz[2];
            int8_t y_angle, uv_angle, cfl_alpha[2];
        }; // intra
        struct {
            int8_t ref[2];
            uint8_t comp_type, wedge_idx, mask_sign;
            uint8_t inter_mode, drl_idx;
            uint8_t interintra_type, interintra_mode;
            uint8_t motion_mode;
            uint8_t max_ytx, filter2d;
            uint16_t tx_split[2];
            mv mv[2];
        }; // inter
    };
} Av1Block;
```

+ palette, palette indices, transform type, transform coefficients

- Each tile (row/col) in pass 1 is completely independent and can run in its own thread
- In theory, we could signal completion of each individual tile_sbrow (as a bitmask) so that subsequent threads (for ref_mvs or seg_ids) could wait on that tile-independently
    - However, ATM, we only signal sbrow completion linearly
- After pass 1, entropy context signaling causes subsequent frame threads that depend on this entropy context to be woken up so their pass 1 can start

High-level overview of decoding process (4)

`decode_b()`, pass != 1, reconstruction



- Each tile_col is independent and can run in its own thread
  - For each inter block, we wait for the reference frame thread to have completed reconstruction of that sbrow
- After completion of each sbrow, we signal the main frame thread to process the next postfilter_sbrow
  - Postfilter is not yet threaded, but we may add that later if it has merit
- The main thread then signals progression of block reconstruction to any subsequent frame waiting for completion of this sbrow

Next steps

RELEASE
TODAY

Next steps for dav1d:

- Finish everything
- 12 bits/component
- Film grain
- SIMD
- More platforms
- Unit tests, fuzzing

Features

- 8+10 bits/component
- All bitstream tools
- Fast, small, efficient
- Multi-threaded (tile + frame)

We need you!

We need you to contribute!

- Code
  - SIMD
  - Platforms ports
  - Tools & Bindings
- Use it!
  - In your app
- Contribute/donate!

Questions?

Thanks to the Alliance for Open Media for sponsoring this work!